Sourcegraph

How DevOps teams can build a data-driven relationship with your codebase

With strong DevOps, your code reaches customers faster and more smoothly, you can trust your automated systems for testing and security, and your teams have more time to focus on the highest-leverage aspects of their roles.

But getting there isn't simple. While advanced tooling and best practices are widely available, it's still too hard to answer questions like:

- How many teams are actually using the tooling we've built?
- How many services are configured using our security and deployment best practices
- What teams are deviating from the "paved road" paths we offer?
  (Why, and how should that inform what we choose to provide?)
- Can we safely deprecate this system we don't use anymore? How long will that take?

Right now, to answer these questions you either spend hours trying to track the information manually, surveying other teams, or building one-off scripts. Or you give up.

DevOps teams should instead build the same data-driven relationship to their code that they expect from their other engineering systems. It's one thing to implement and connect important DevOps systems – it's equally important to know the how, if, where, and why about the actual impact of your DevOps work.

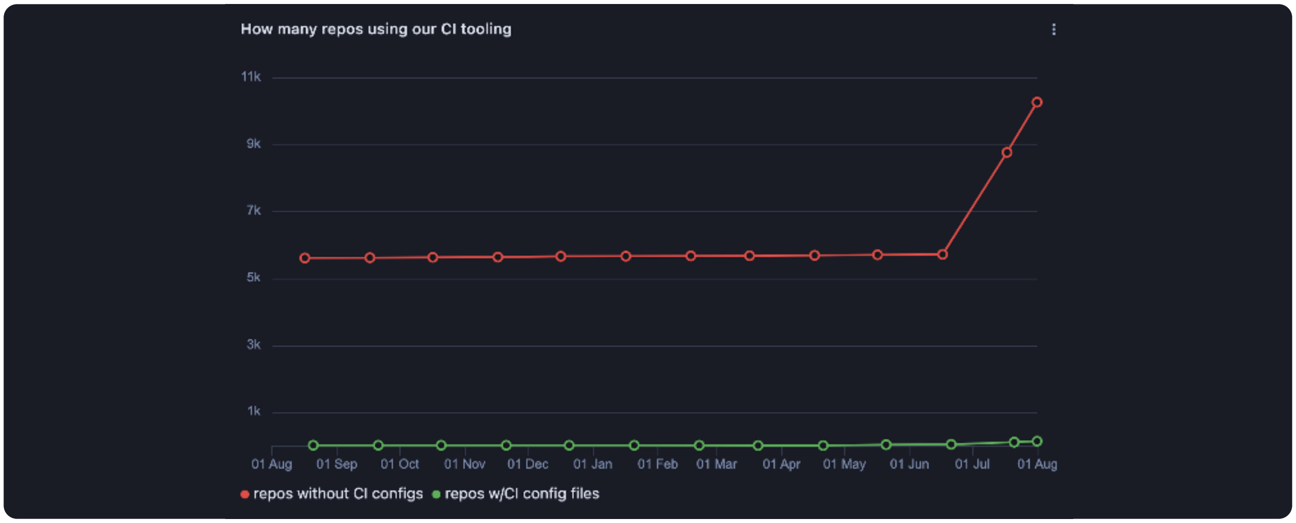
Luckily, those signals all exist in your codebase – you just need to surface them.

Data-driven tracking of DevOps practices and standards

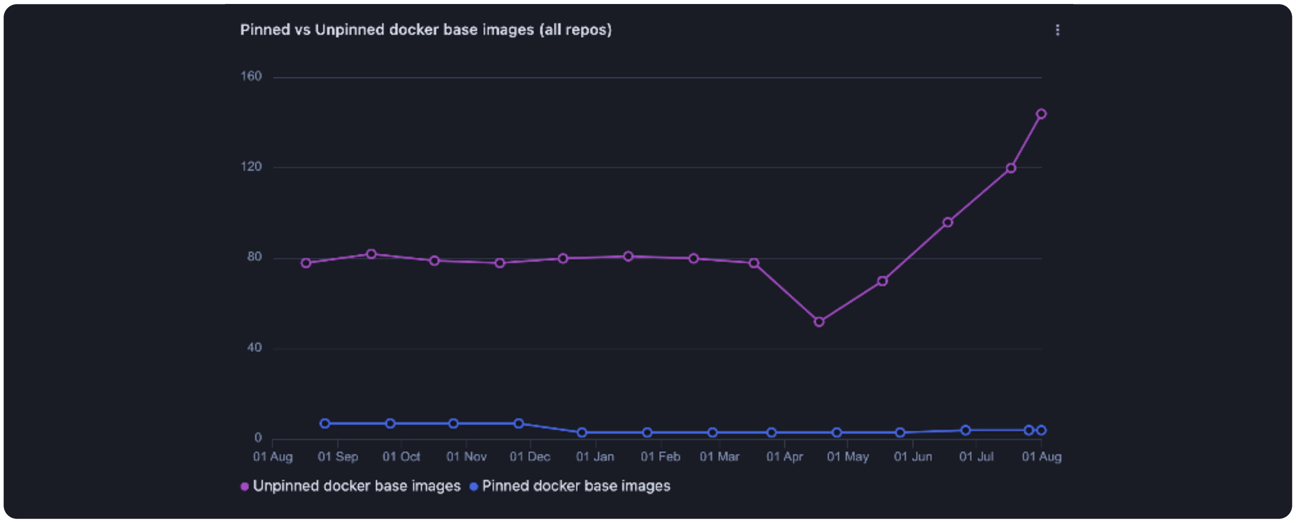
Just offering CI, CD, security, and other deployment systems and standards to engineering teams isn’t enough if the tools aren’t used.

But tracking the usage is possible directly from the codebase. For example, you can track:

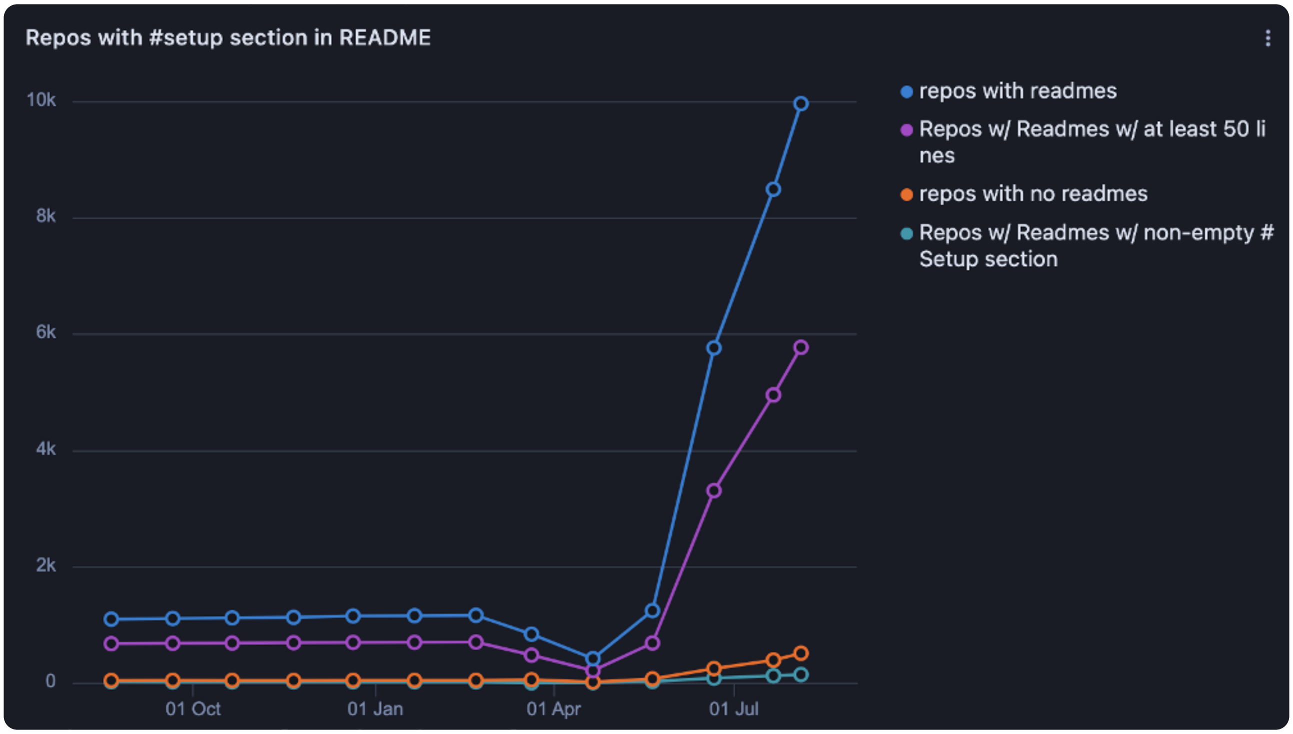
Whether all new repos and microservices are connected to your CI pipeline:



If docker images are using unpinned or pinned versions



Whether readmes include deploy instructions or mention of a specific process within their #deploy or #setup section:



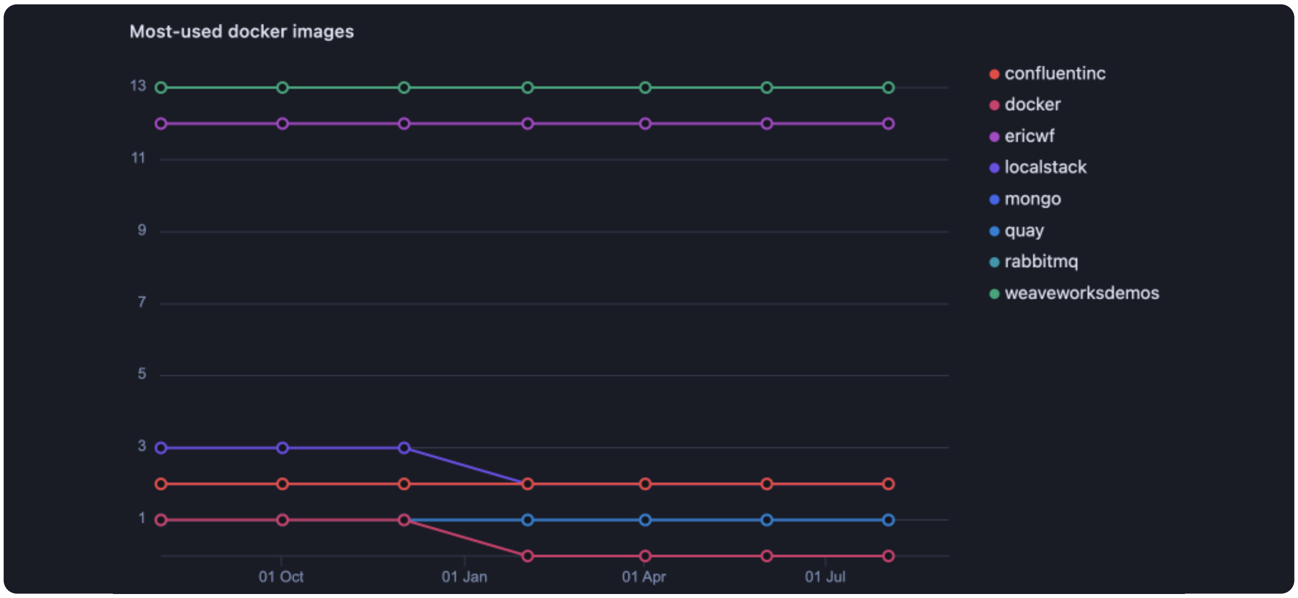
Having this information tells you **whether or not your DevOps practices are adopted** across your organization:

1. **Discover teams and projects that aren't on the paved path, and understand why**. If you can uncover that teams have stopped using the standard modules or config files, you can then follow up with those teams directly to decide whether it's a situation where the paved path is still best or whether a new standard configuration should be supported.
2. **Plan and prioritize future work**. Do teams need better tooling to keep modules up to date? Are systems too hard to set up, and that's why teams aren't adopting them? You should first start by uncovering the most impactful improvements to be made based on existing usage data.
3. **Track the rollout of new compliance, testing, and security systems**. A CI system, linter, or deployment cadence will only have impact if it's actually adopted. When you plan and assess your work, you need to know whether it's been used across the organization and how long that adoption takes.

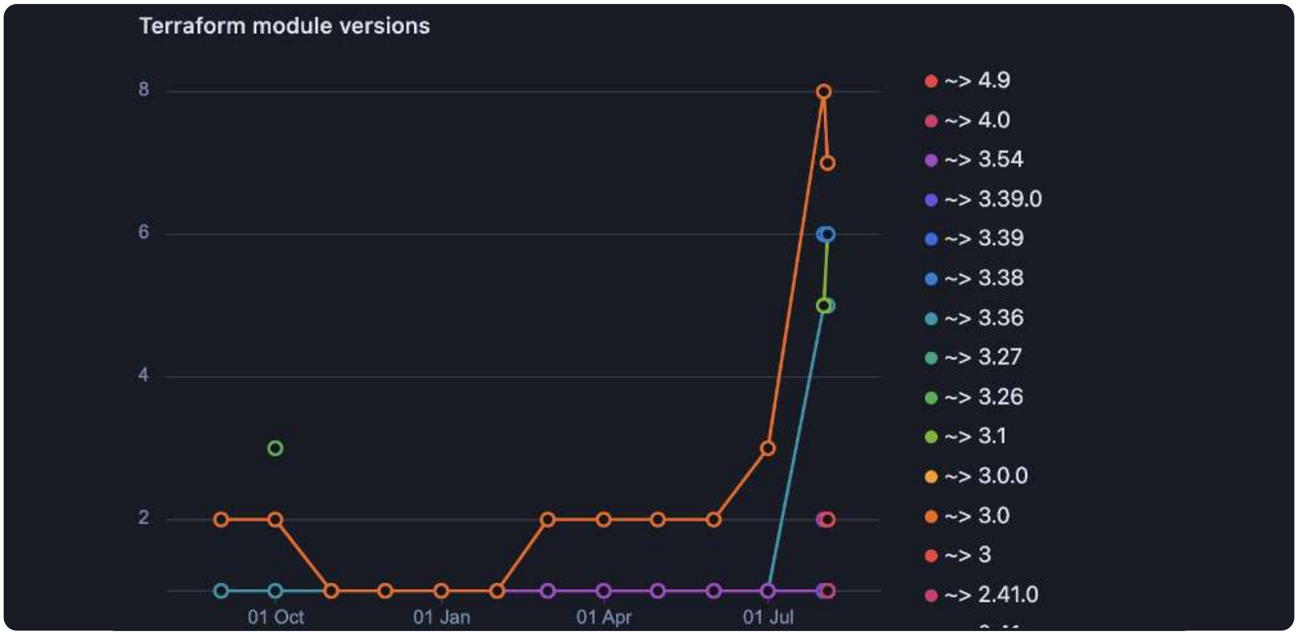
Data-driven configuration monitoring

Infrastructure-as-code means you can monitor infrastructure configurations and changes directly in your codebase, such as:

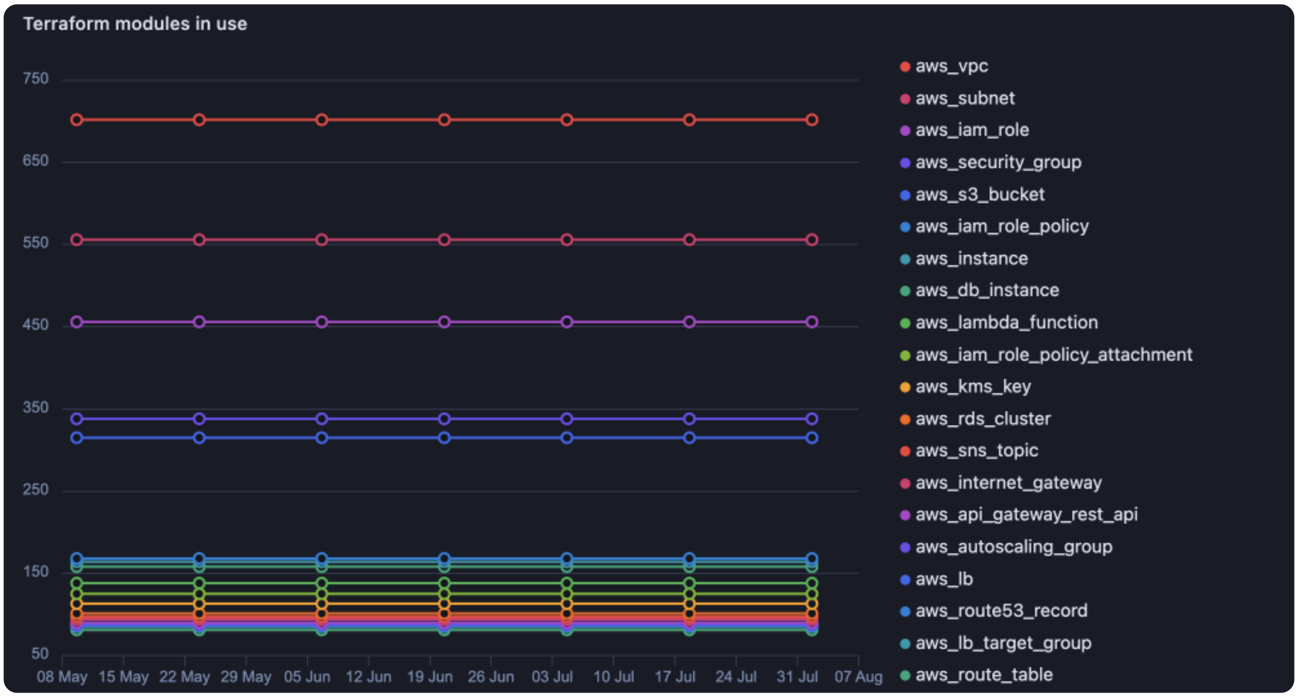
Which Docker images are in use:



Whether Terraform versions are kept up to date:



Which Terraform modules are most popular:



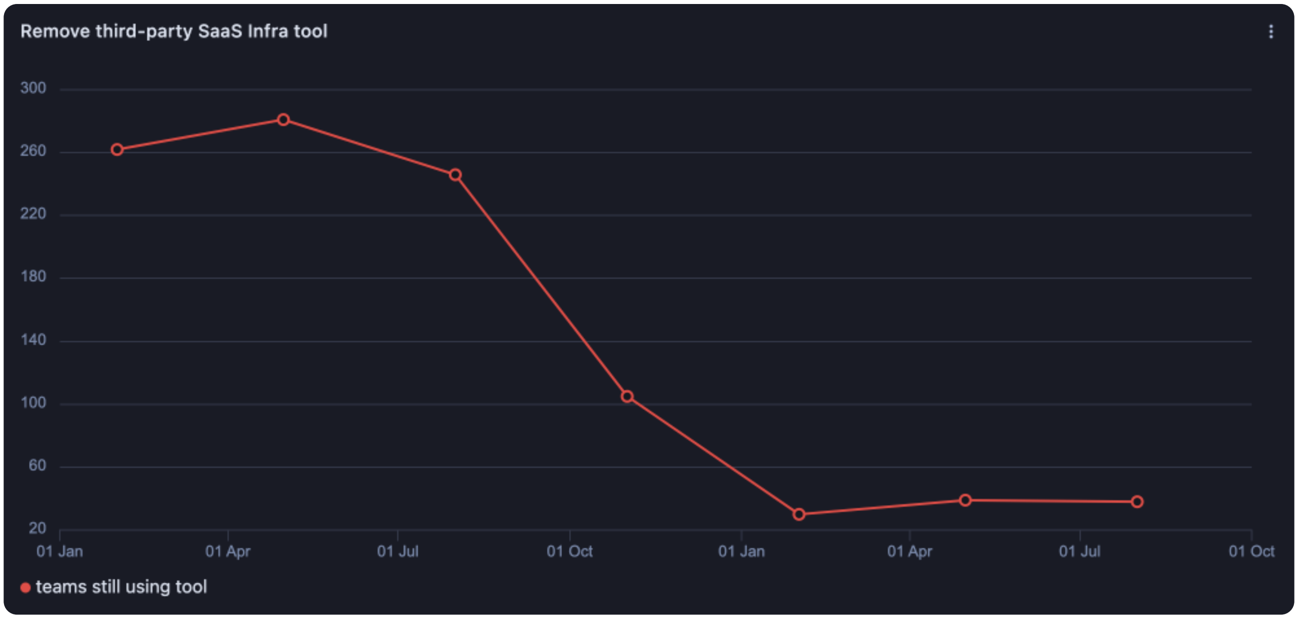
To maintain your organization’s DevOps health, you should:

1. **Keep software versions up to date and prevent vulnerabilities**. Determine how many sub-versions you’ll support and ensure teams have visibility into that KPI. Drill into the data to surface which repos need attention.
2. **Know which configurations to prioritize support, updates, and documentation for**. To empower individual teams and services, you should focus on the most-used infrastructure first.
3. **Surface opportunities for consolidation or cost reduction**. If you have multiple services in use that have the same function, or a service that’s notably more costly to deploy, you can unify folks onto the most-used service or a more cost-optimized service, and save time and money in support and deployment costs.
4. **Be confident in the source of truth**. Rather than hoping everyone remembered to update a spreadsheet manually, just look in the code itself.

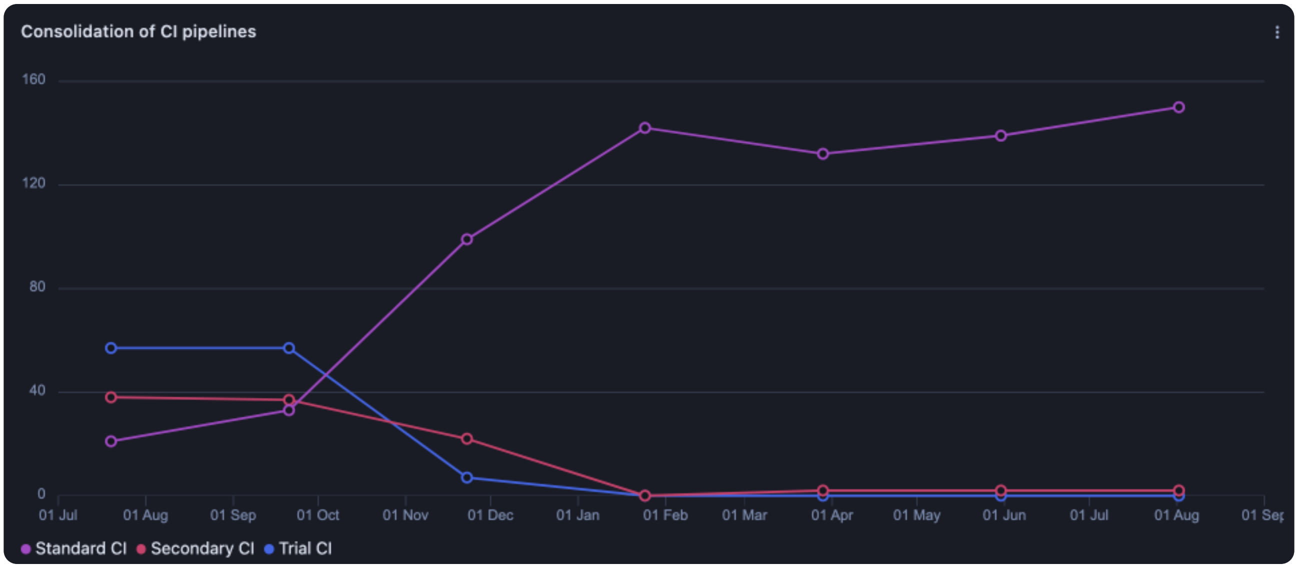
Data-driven migrations and deprecations

When you make major changes to your tooling, you need an accurate picture of progress to date. For example:

Track removal of a third-party DevOps SaaS you no longer use:



Ensure everyone migrates to the supported CI system:



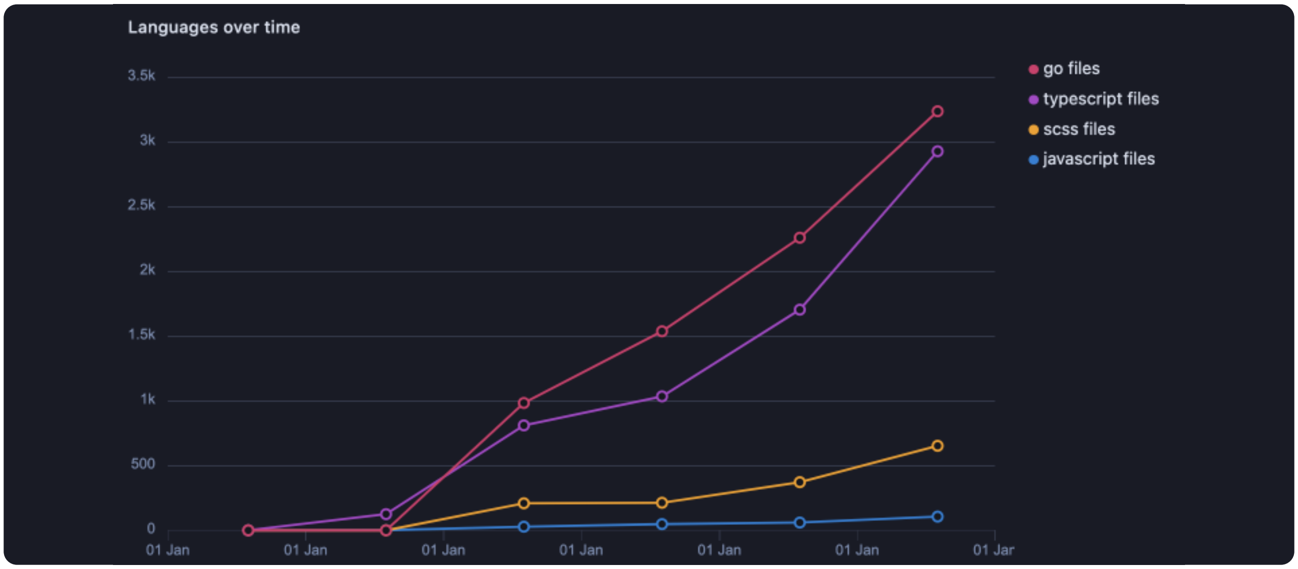
Having this tracking will **accelerate** the progress of your migration:

1. **Immediately discover and fix slowdowns**. If the project starts to stall out, you can filter the data by repository or team area and understand what's going on, then fix it. Without this visibility, you might lose weeks thinking things are going well when they aren't.
2. **Ensure deprecations get done completely (and save money)**. Often, deprecations stall out at 70-80% completion because people aren't able to track the total progress. Delays rapidly become costly if you're deprecating build or deployment SaaS tools you have to pay for until they're fully removed.
3. **Surface which teams and what code gets migrated fastest, and learn from what works**. The inverse of finding slowdowns is that you can also learn from the progress individual teams make.
4. **Motivate teammates and teams across the organization**. It's difficult to get buy-in on cross-team projects, but having a chart to burn up provides a central point of focus and excitement as it progresses.

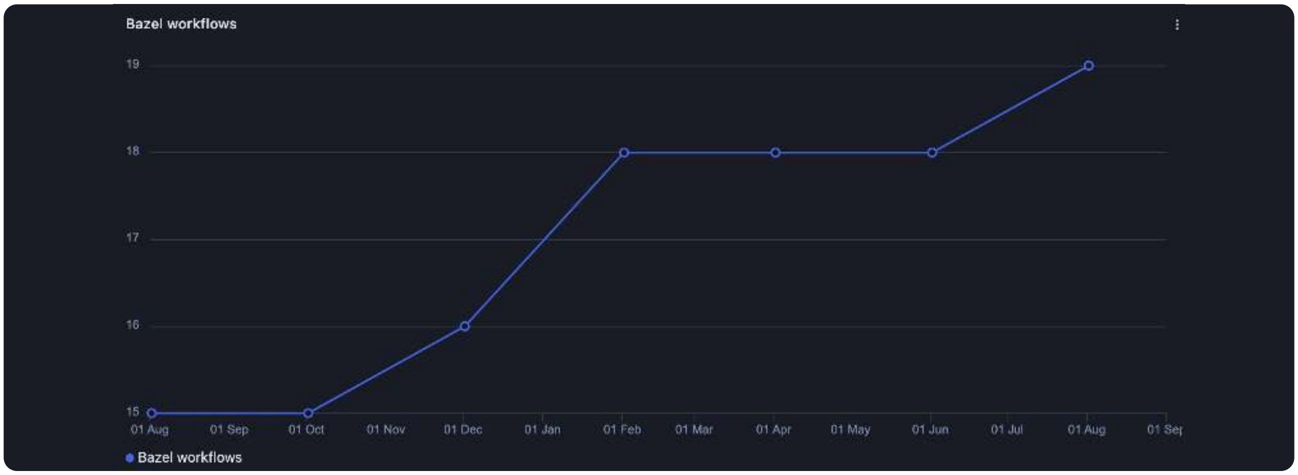
Data-driven trend analysis to future-proof your DevOps

The best DevOps organizations get ahead of their engineering teams' future needs. To do so, you need to understand the trends across your codebase, like:

Which languages and associated tooling needs will grow fastest:



How many projects are experimenting with Bazel and is that momentum accelerating:



Getting ahead of trends allows you to **be prepared for the growth of your organization**:

1. **Ensure you have systems and people in place for the fastest-growing areas.** Never get taken by surprise that you should have prioritized Java tooling or hired more people familiar with Ruby-specific DevOps tools.
2. **Guide new initiatives from the start.** Rather than jumping in to give a late yes/no decision on whether a new practice is acceptable, watch early trends and understand where you should provide guidance, support, or guard rails.

Being a data-driven DevOps team gives you a source of truth you can use to execute, plan, and support your key DevOps goals

Every DevOps team has dealt with a slow tool rollout, an improper configuration causing confusion and bugs, or a lack of visibility on the full impact of your work.

Few of them have access to data that would make DevOps more consistent, secure, and proactive.

Let’s build a world in which data-driven DevOps is the standard. Your DevOps team should not waste time fighting fires, trying to figure out the impact of their work, what they should do next, or whether a new initiative will actually support the future needs of your organization. They should be preventing fires, doing proactive testing, and guiding the engineering team.

Sourcegraph

about.sourcegraph.com

Sourcegraph is a code intelligence platform that unlocks developer velocity by helping engineering teams understand, fix, and automate across their entire codebase.